시뮬레이터 기본 교육 및 Control & Planning 알고리즘 개발

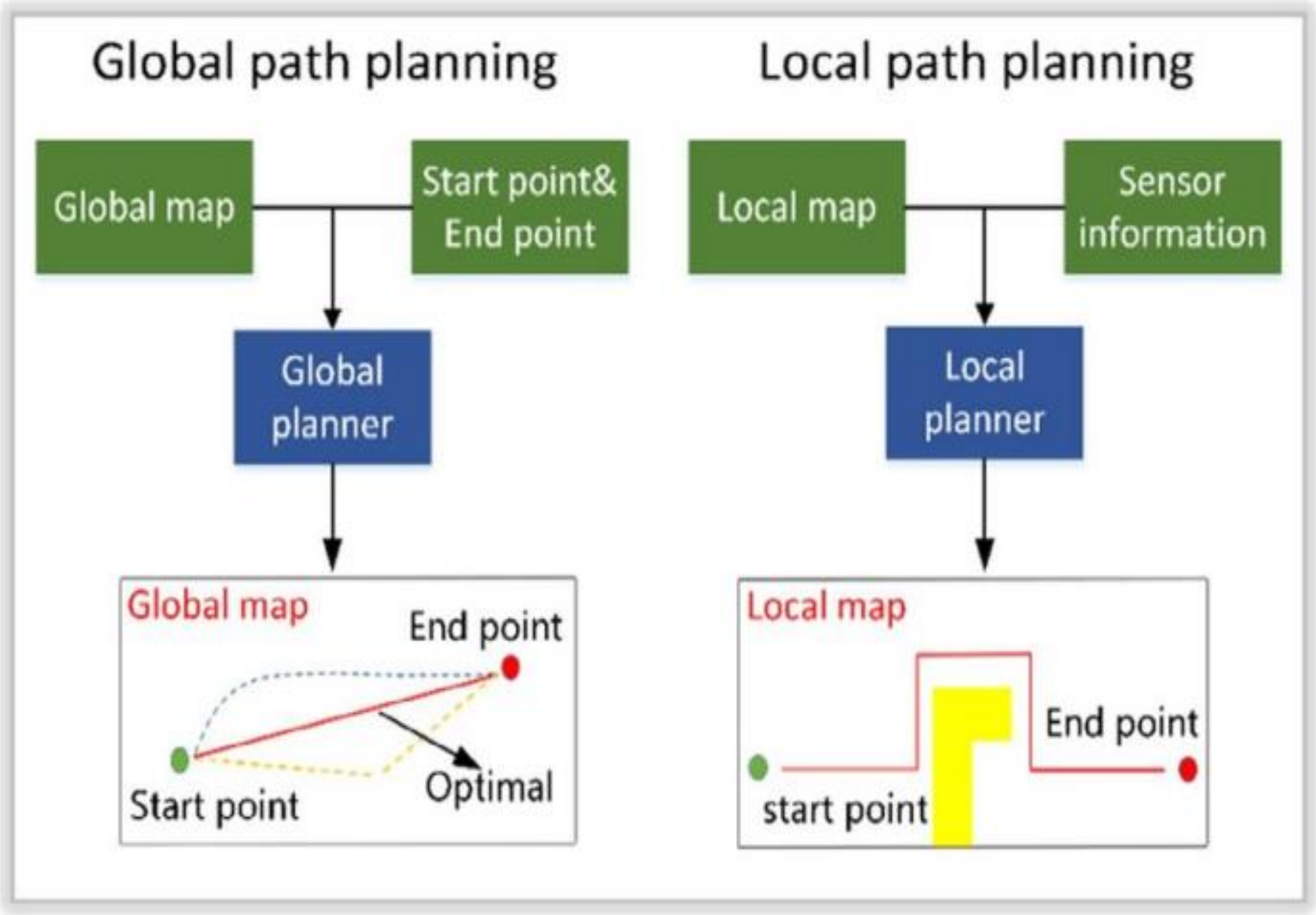
프로젝트 지향 자율주행차 전문인력 양성과정

목차

1. 경로 생성

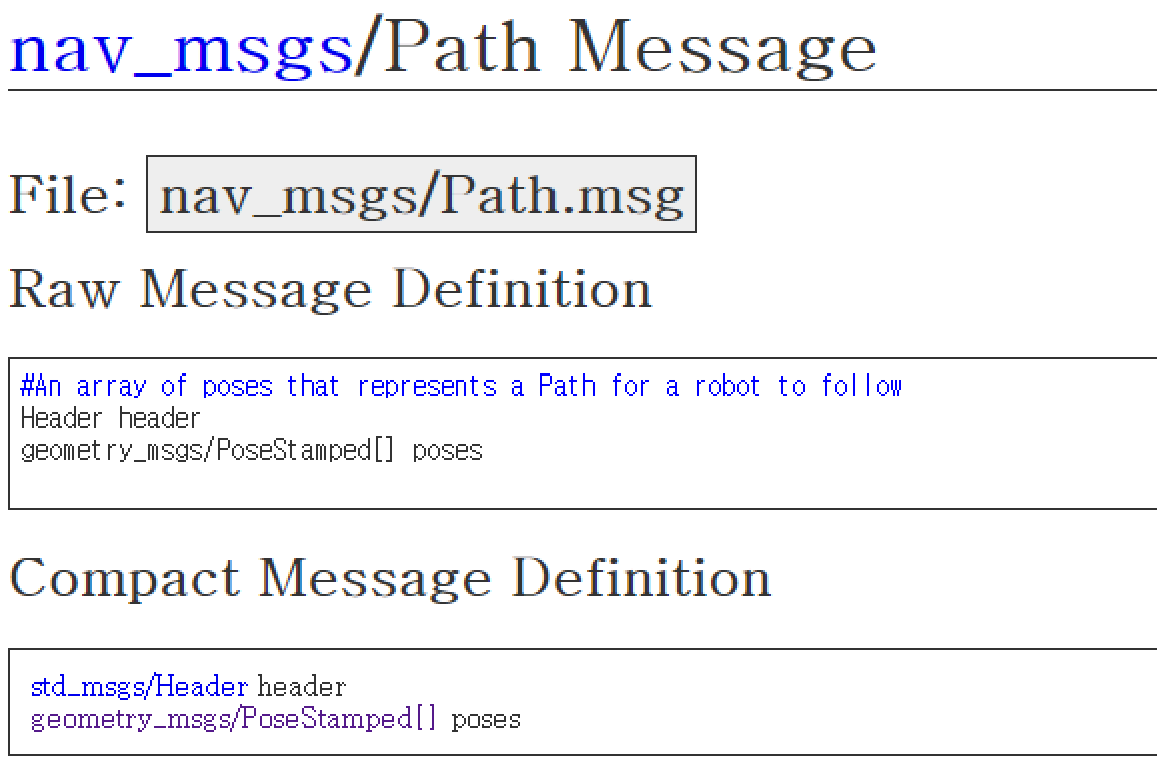
경로 생성

- **경로 생성**
  - **전역(Global) 경로, 지역(Local)경로**



경로 생성

- **경로 생성**
  - **Odometry를 이용해서 주행했던 경로를 텍스트로 저장**



nav_msgs/Path Message

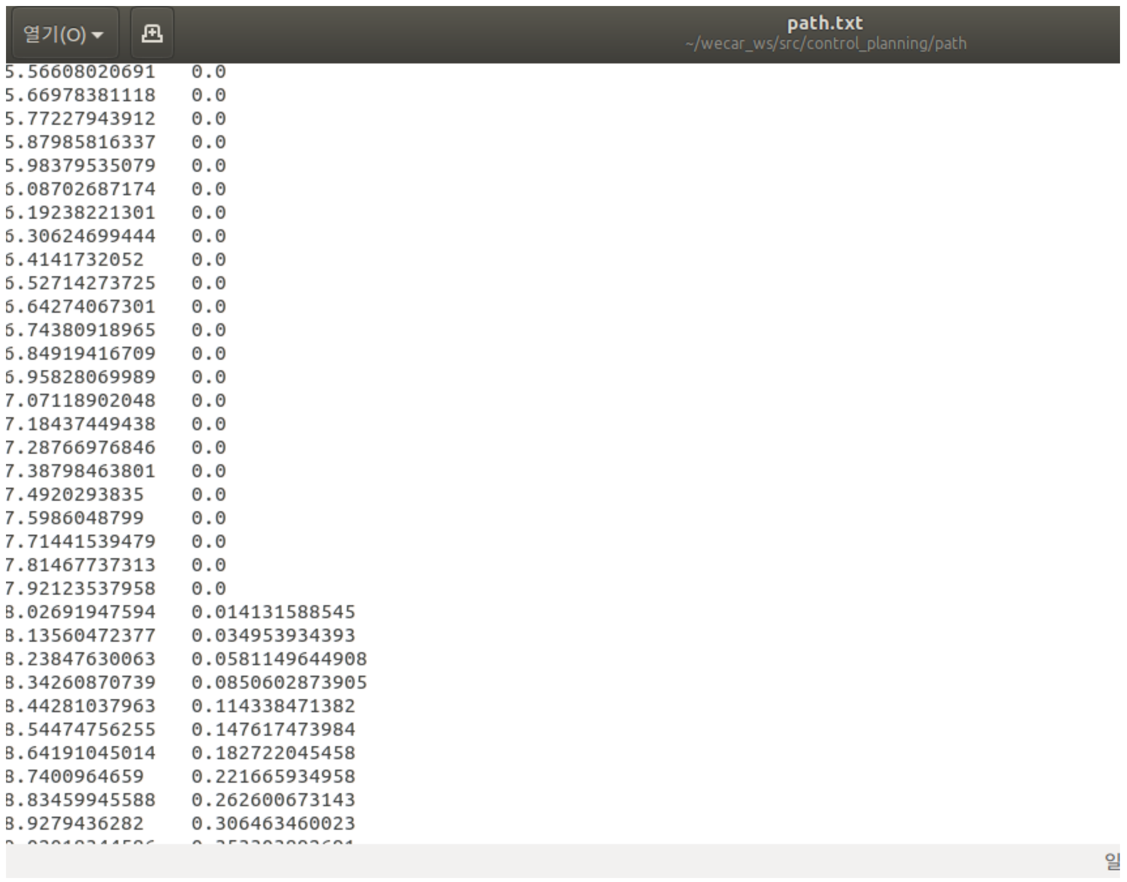
File: nav_msgs/Path.msg

Raw Message Definition

```
#An array of poses that represents a Path for a robot to follow
Header header
geometry_msgs/PoseStamped[] poses
```

Compact Message Definition

```
std_msgs/Header header
geometry_msgs/PoseStamped[] poses
```

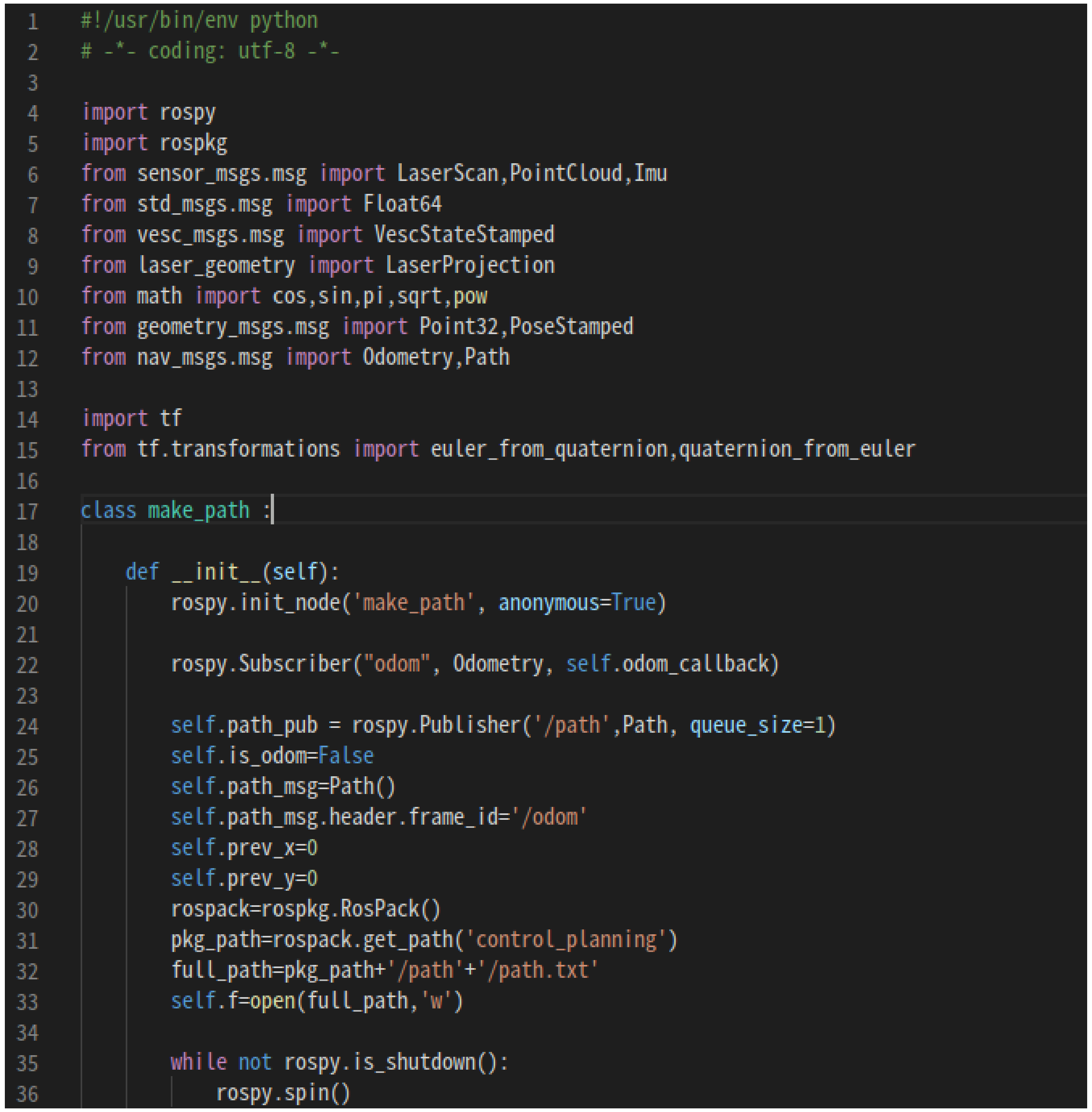


```
5.56608020691   0.0
5.66978381118   0.0
5.77227943912   0.0
5.87985816337   0.0
5.98379535079   0.0
6.08702687174   0.0
6.19238221301   0.0
6.30624699444   0.0
6.4141732052    0.0
6.52714273725   0.0
6.64274067301   0.0
6.74380918965   0.0
6.84919416709   0.0
6.95828069989   0.0
7.07118902048   0.0
7.18437449438   0.0
7.28766976846   0.0
7.38798463801   0.0
7.4920293835    0.0
7.5986048799    0.0
7.71441539479   0.0
7.81467737313   0.0
7.92123537958   0.0
8.02691947594   0.014131588545
8.13560472377   0.034953934393
8.23847630063   0.0581149644908
8.34260870739   0.0850602873905
8.44281037963   0.114338471382
8.54474756255   0.147617473984
8.64191045014   0.182722045458
8.7400964659    0.221665934958
8.83459945588   0.262600673143
8.9279436282    0.306463460023
```

http://docs.ros.org/melodic/api/nav_msgs/html/msg/Path.html

경로 생성

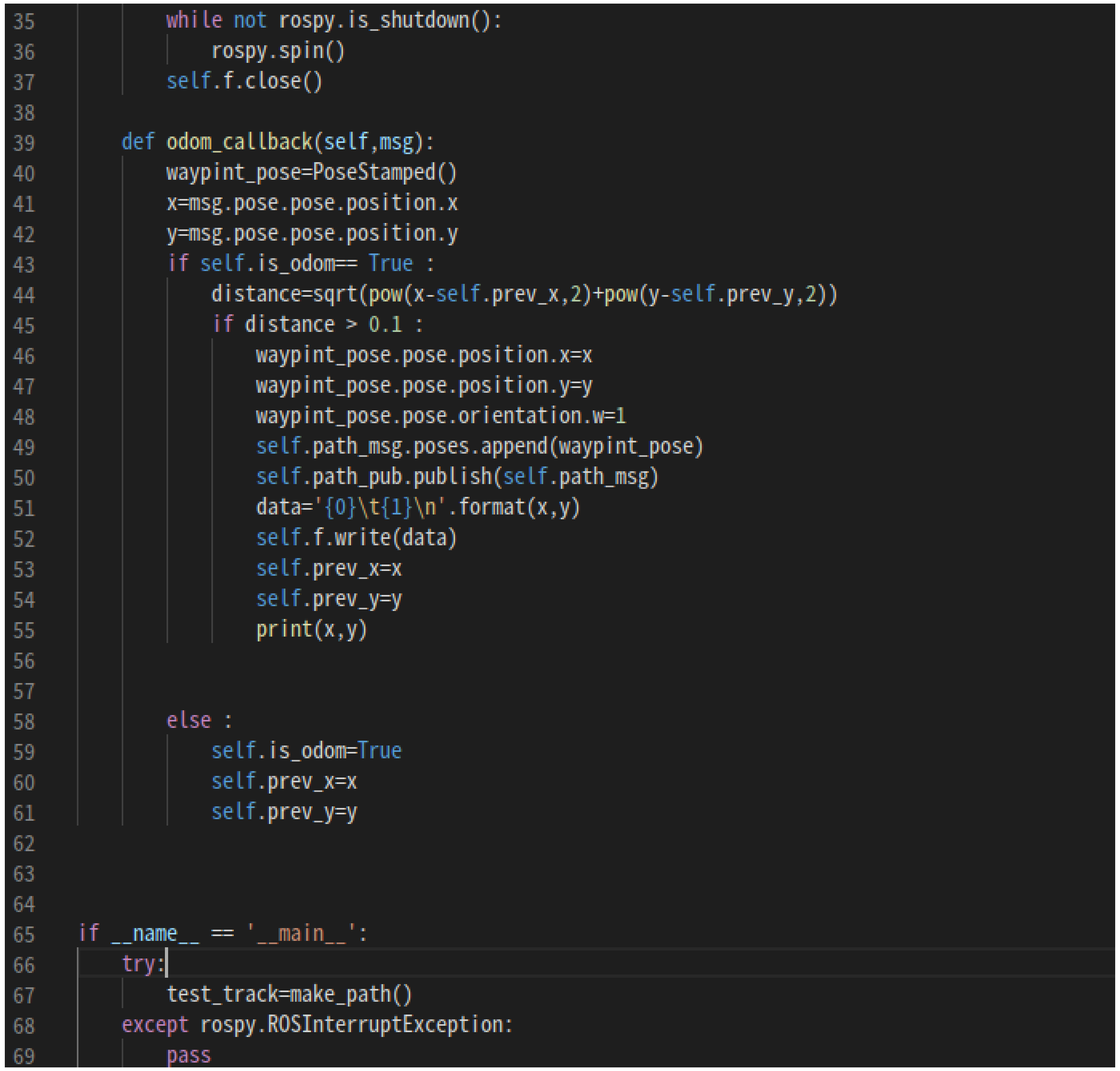
- **경로 생성**



```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import rospkg
from sensor_msgs.msg import LaserScan,PointCloud,Imu
from std_msgs.msg import Float64
from vesc_msgs.msg import VescStateStamped
from laser_geometry import LaserProjection
from math import cos,sin,pi,sqrt,pow
from geometry_msgs.msg import Point32,PoseStamped
from nav_msgs.msg import Odometry,Path

import tf
from tf.transformations import euler_from_quaternion,quaternion_from_euler

class make_path :

    def __init__(self):
        rospy.init_node('make_path', anonymous=True)

        rospy.Subscriber("odom", Odometry, self.odom_callback)

        self.path_pub = rospy.Publisher('/path',Path, queue_size=1)
        self.is_odom=False
        self.path_msg=Path()
        self.path_msg.header.frame_id='/odom'
        self.prev_x=0
        self.prev_y=0
        rospack=rospkg.RosPack()
        pkg_path=rospack.get_path('control_planning')
        full_path=pkg_path+'/path'+'/path.txt'
        self.f=open(full_path,'w')

        while not rospy.is_shutdown():
            rospy.spin()
```

경로 생성

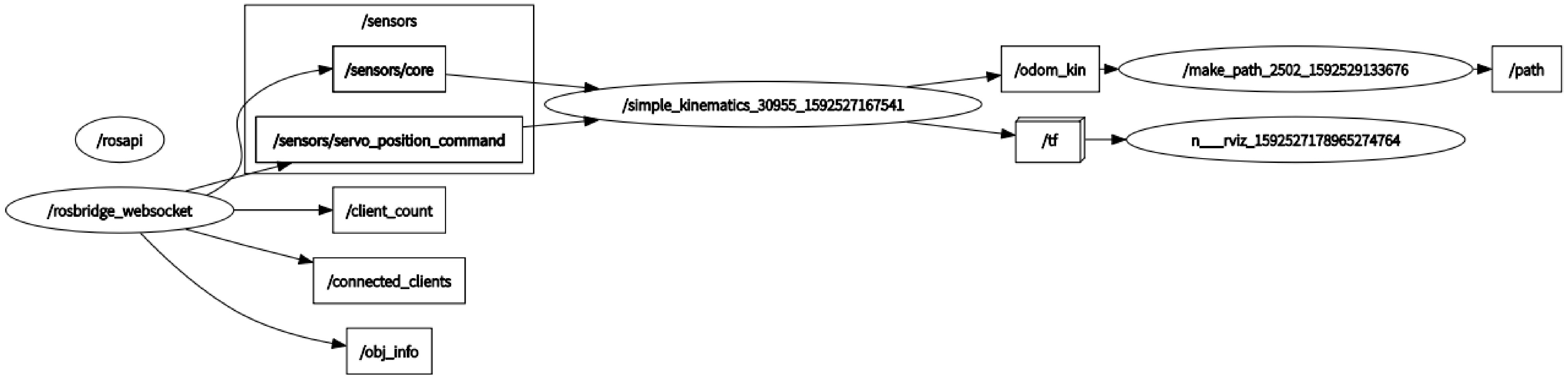
- **경로 생성**



```python
        while not rospy.is_shutdown():
            rospy.spin()
        self.f.close()

    def odom_callback(self,msg):
        waypint_pose=PoseStamped()
        x=msg.pose.pose.position.x
        y=msg.pose.pose.position.y
        if self.is_odom== True :
            distance=sqrt(pow(x-self.prev_x,2)+pow(y-self.prev_y,2))
            if distance > 0.1 :
                waypint_pose.pose.position.x=x
                waypint_pose.pose.position.y=y
                waypint_pose.pose.orientation.w=1
                self.path_msg.poses.append(waypint_pose)
                self.path_pub.publish(self.path_msg)
                data='{0}\t{1}\n'.format(x,y)
                self.f.write(data)
                self.prev_x=x
                self.prev_y=y
                print(x,y)


        else :
            self.is_odom=True
            self.prev_x=x
            self.prev_y=y



if __name__ == '__main__':
    try:
        test_track=make_path()
    except rospy.ROSInterruptException:
        pass
```

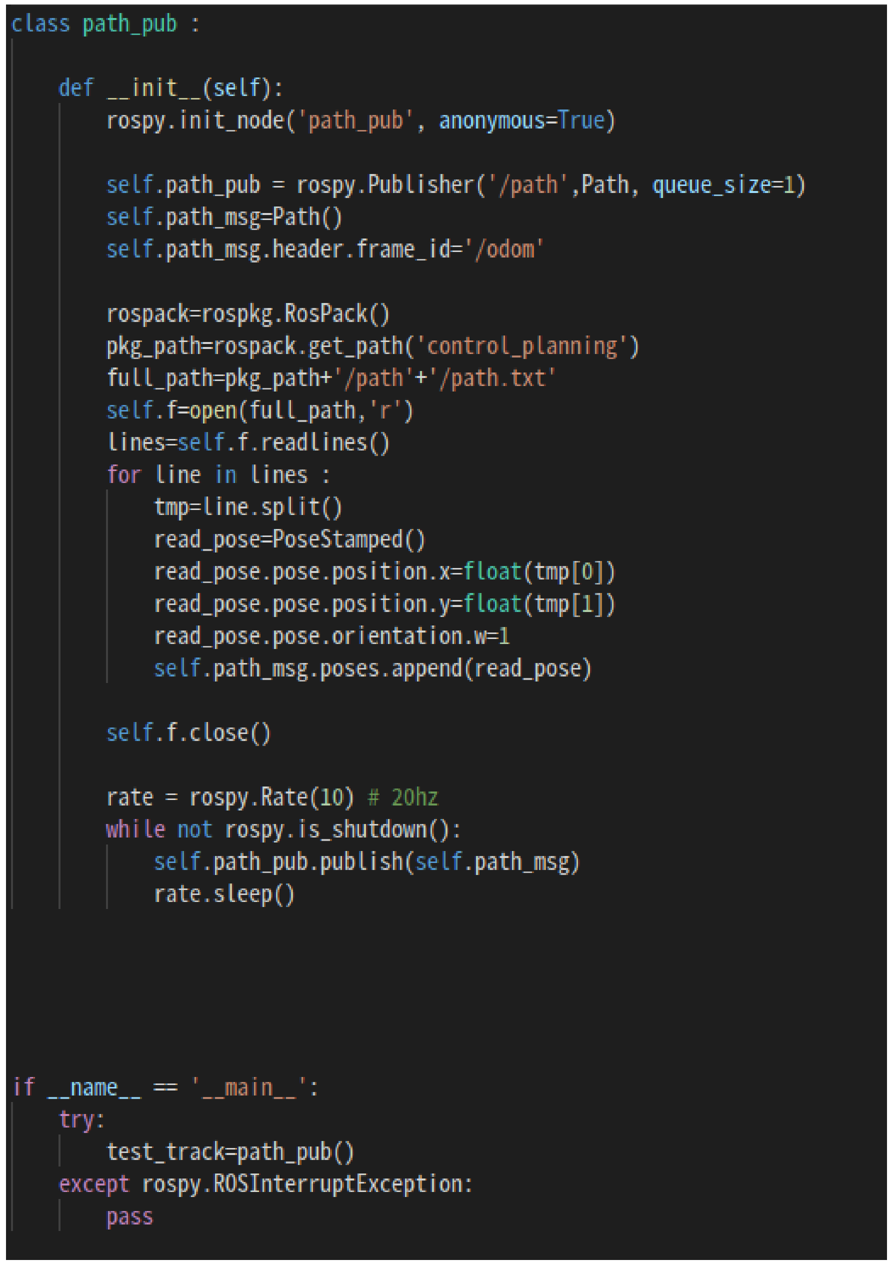
경로 생성

- 경로 생성
  - rqt_graph 모습


/sensors
/sensors/core
/sensors/servo_position_command
/rosapi
/rosbridge_websocket
/client_count
/connected_clients
/obj_info
/simple_kinematics_30955_1592527167541
/odom_kin
/make_path_2502_1592529133676
/path
/tf
n___rviz_1592527178965274764

경로 생성

- 경로 읽어오기
  - 저장한 경로를 읽어서 path 라는 토픽으로 Publish한다.



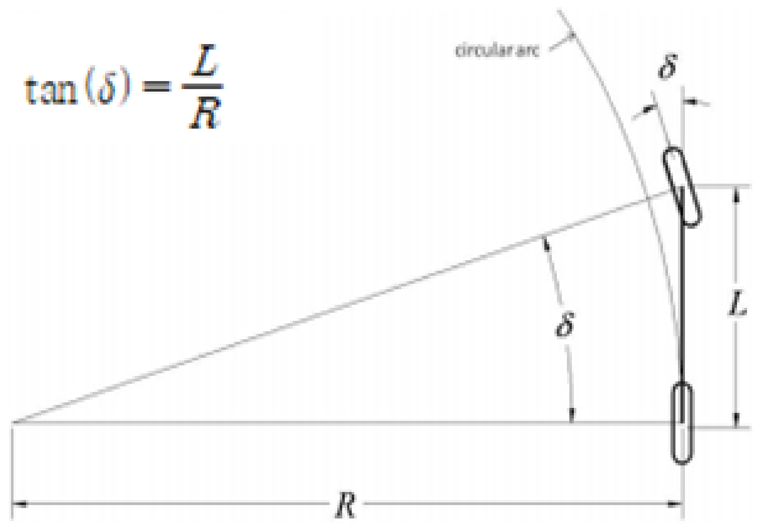
```
class path_pub :

    def __init__(self):
        rospy.init_node('path_pub', anonymous=True)

        self.path_pub = rospy.Publisher('/path',Path, queue_size=1)
        self.path_msg=Path()
        self.path_msg.header.frame_id='/odom'

        rospack=rospkg.RosPack()
        pkg_path=rospack.get_path('control_planning')
        full_path=pkg_path+'/path'+'/path.txt'
        self.f=open(full_path,'r')
        lines=self.f.readlines()
        for line in lines :
            tmp=line.split()
            read_pose=PoseStamped()
            read_pose.pose.position.x=float(tmp[0])
            read_pose.pose.position.y=float(tmp[1])
            read_pose.pose.orientation.w=1
            self.path_msg.poses.append(read_pose)

        self.f.close()

        rate = rospy.Rate(10) # 20hz
        while not rospy.is_shutdown():
            self.path_pub.publish(self.path_msg)
            rate.sleep()



if __name__ == '__main__':
    try:
        test_track=path_pub()
    except rospy.ROSInterruptException:
        pass
```

2. 경로 추종

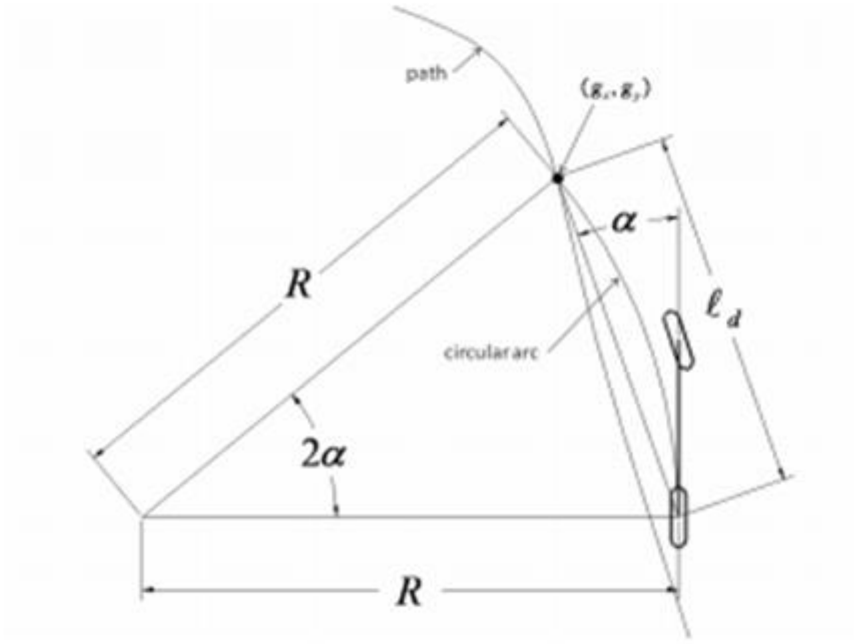
경로 추종

- Pure pursuit
  - Pure Pursuit은 경로 위의 한 점을 원 호를 그리며 따라가는 경로 추종 알고리즘
  - 자동차의 기구학과 전방주시거리(Look-Ahead-Distance)라는 하나의 파라미터만 가지고 조향각을 간단하게 계산할 수 있다
  - Pure Pursuit에서는 실제 자동차 모델(Ackermann geometry)을 단순화 한 Bicycle 모델사용

$$\tan(\delta) = \frac{L}{R}$$



<Bicycle Geometry>



<Pure Pursuit Geometry>

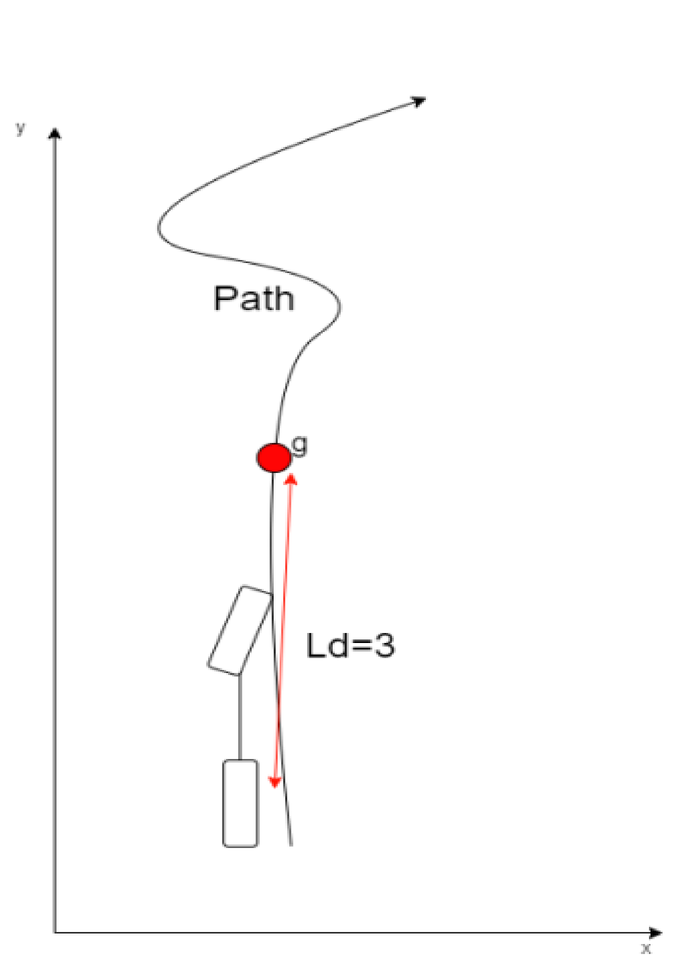
$$l_d \cos(\alpha) = R\sin(2\alpha)$$
$$\frac{l_d}{2\sin(\alpha)\cos(\alpha)} = \frac{R}{\cos(\alpha)}$$
$$\frac{l_d}{\sin(\alpha)} = 2R$$
$$\frac{1}{R} = \frac{2\sin(\alpha)}{l_d}$$
$$\delta = \tan^{-1}\left(\frac{L}{R}\right)$$
$$\delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{l_d}\right)$$

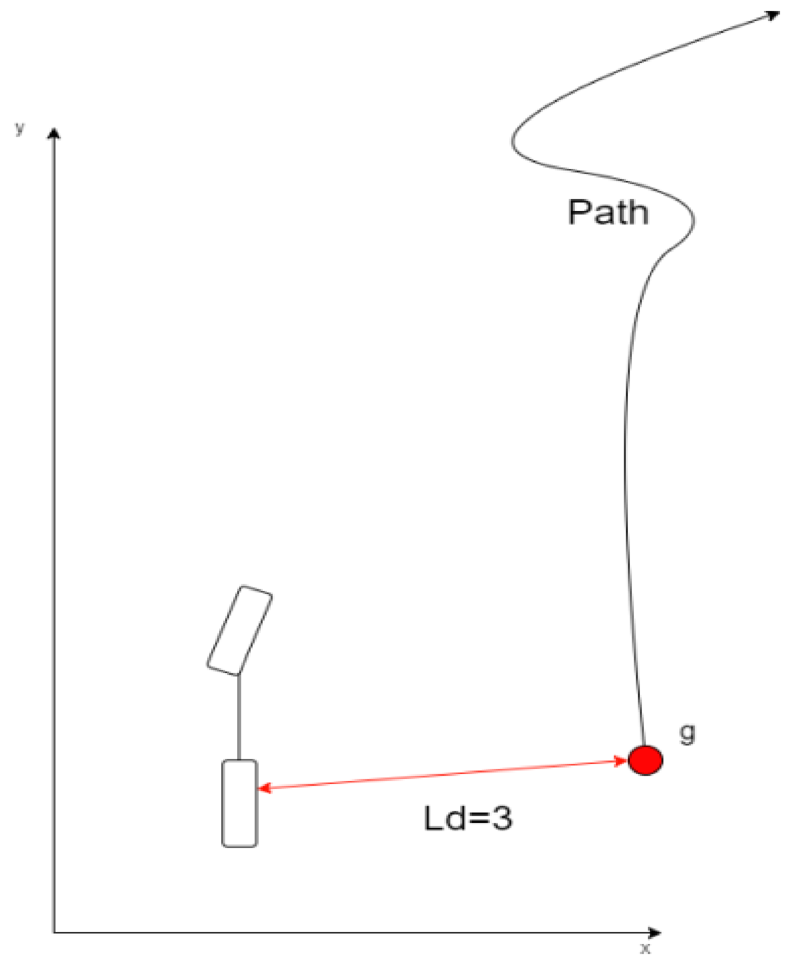
<조향각 유도식>

https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html

경로 추종

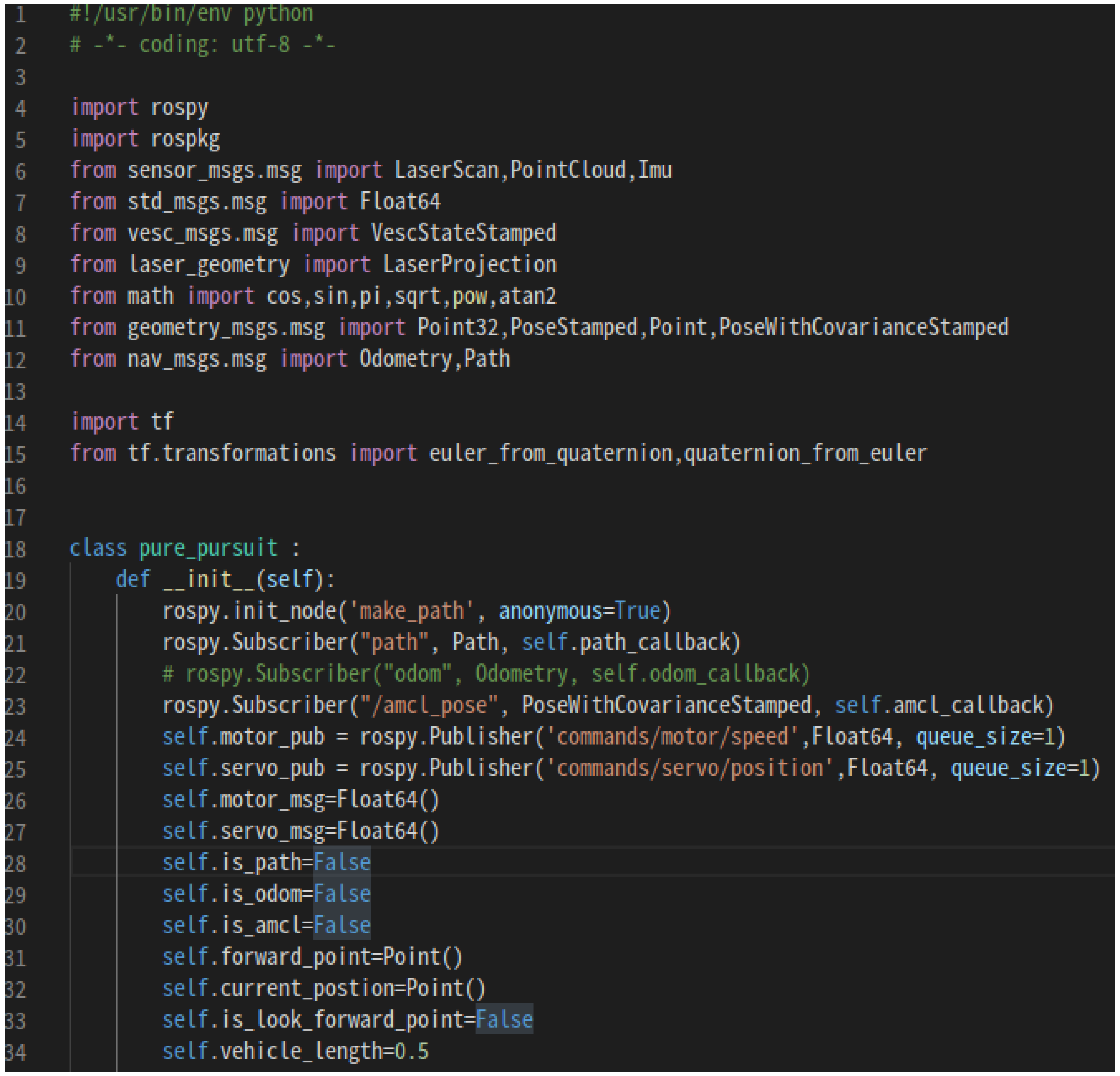
- **Pure pursuit**
  - **고려할 사항들**





경로 추종

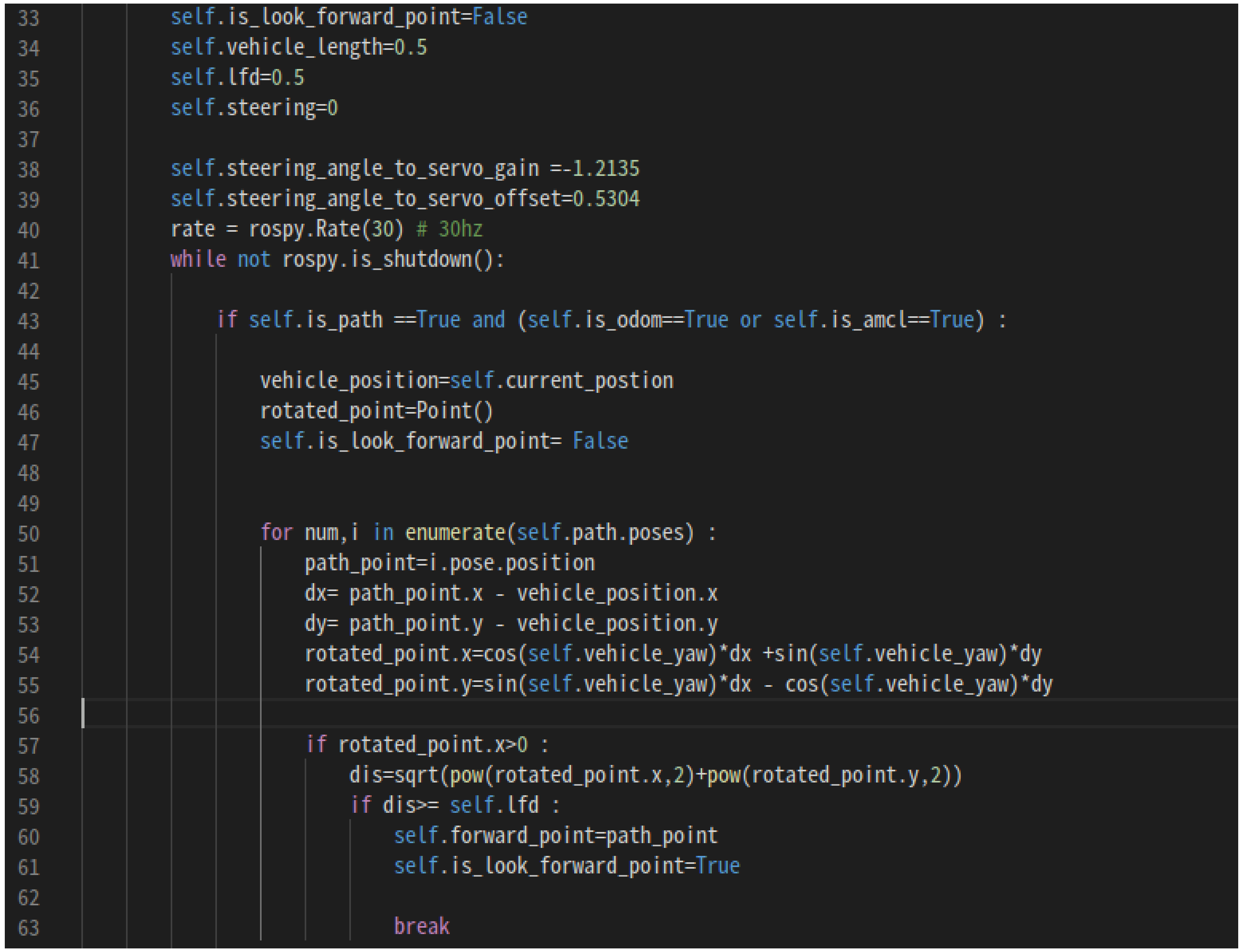
- **Pure pursuit**



```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import rospkg
from sensor_msgs.msg import LaserScan,PointCloud,Imu
from std_msgs.msg import Float64
from vesc_msgs.msg import VescStateStamped
from laser_geometry import LaserProjection
from math import cos,sin,pi,sqrt,pow,atan2
from geometry_msgs.msg import Point32,PoseStamped,Point,PoseWithCovarianceStamped
from nav_msgs.msg import Odometry,Path

import tf
from tf.transformations import euler_from_quaternion,quaternion_from_euler


class pure_pursuit :
    def __init__(self):
        rospy.init_node('make_path', anonymous=True)
        rospy.Subscriber("path", Path, self.path_callback)
        # rospy.Subscriber("odom", Odometry, self.odom_callback)
        rospy.Subscriber("/amcl_pose", PoseWithCovarianceStamped, self.amcl_callback)
        self.motor_pub = rospy.Publisher('commands/motor/speed',Float64, queue_size=1)
        self.servo_pub = rospy.Publisher('commands/servo/position',Float64, queue_size=1)
        self.motor_msg=Float64()
        self.servo_msg=Float64()
        self.is_path=False
        self.is_odom=False
        self.is_amcl=False
        self.forward_point=Point()
        self.current_postion=Point()
        self.is_look_forward_point=False
        self.vehicle_length=0.5
```

경로 추종

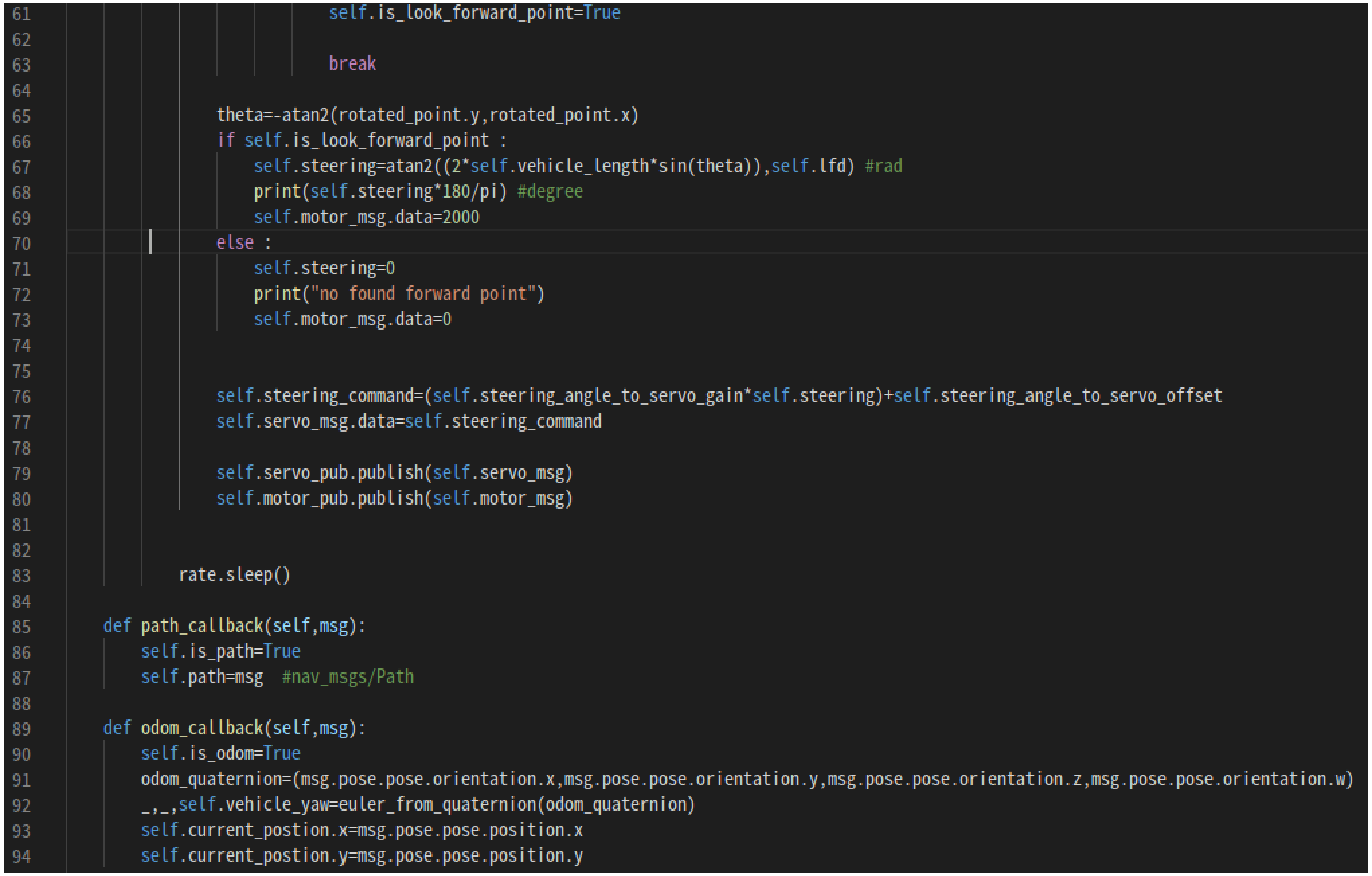
- **Pure pursuit**



```
        self.is_look_forward_point=False
        self.vehicle_length=0.5
        self.lfd=0.5
        self.steering=0

        self.steering_angle_to_servo_gain =-1.2135
        self.steering_angle_to_servo_offset=0.5304
        rate = rospy.Rate(30) # 30hz
        while not rospy.is_shutdown():

            if self.is_path ==True and (self.is_odom==True or self.is_amcl==True) :

                vehicle_position=self.current_postion
                rotated_point=Point()
                self.is_look_forward_point= False


                for num,i in enumerate(self.path.poses) :
                    path_point=i.pose.position
                    dx= path_point.x - vehicle_position.x
                    dy= path_point.y - vehicle_position.y
                    rotated_point.x=cos(self.vehicle_yaw)*dx +sin(self.vehicle_yaw)*dy
                    rotated_point.y=sin(self.vehicle_yaw)*dx - cos(self.vehicle_yaw)*dy

                    if rotated_point.x>0 :
                        dis=sqrt(pow(rotated_point.x,2)+pow(rotated_point.y,2))
                        if dis>= self.lfd :
                            self.forward_point=path_point
                            self.is_look_forward_point=True

                            break
```

경로 추종

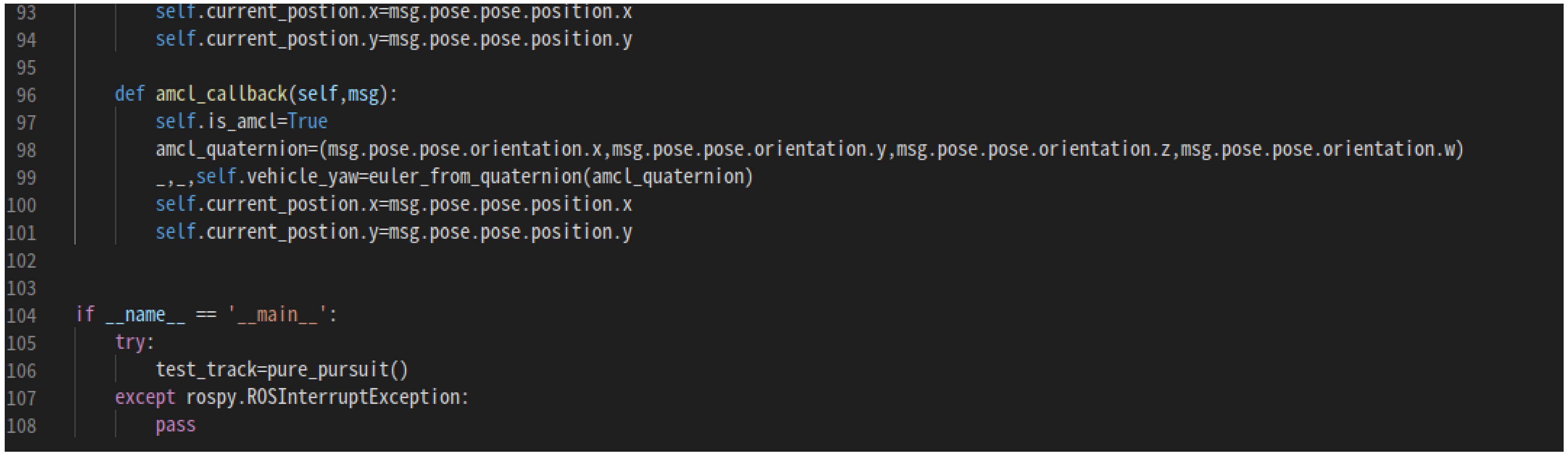
- **Pure pursuit**



```
                            self.is_look_forward_point=True

                            break

                theta=-atan2(rotated_point.y,rotated_point.x)
                if self.is_look_forward_point :
                    self.steering=atan2((2*self.vehicle_length*sin(theta)),self.lfd) #rad
                    print(self.steering*180/pi) #degree
                    self.motor_msg.data=2000
                else :
                    self.steering=0
                    print("no found forward point")
                    self.motor_msg.data=0


                self.steering_command=(self.steering_angle_to_servo_gain*self.steering)+self.steering_angle_to_servo_offset
                self.servo_msg.data=self.steering_command

                self.servo_pub.publish(self.servo_msg)
                self.motor_pub.publish(self.motor_msg)


            rate.sleep()

    def path_callback(self,msg):
        self.is_path=True
        self.path=msg  #nav_msgs/Path

    def odom_callback(self,msg):
        self.is_odom=True
        odom_quaternion=(msg.pose.pose.orientation.x,msg.pose.pose.orientation.y,msg.pose.pose.orientation.z,msg.pose.pose.orientation.w)
        _,_,self.vehicle_yaw=euler_from_quaternion(odom_quaternion)
        self.current_postion.x=msg.pose.pose.position.x
        self.current_postion.y=msg.pose.pose.position.y
```

경로 추종

- **Pure pursuit**



```python
        self.current_postion.x=msg.pose.pose.position.x
        self.current_postion.y=msg.pose.pose.position.y

    def amcl_callback(self,msg):
        self.is_amcl=True
        amcl_quaternion=(msg.pose.pose.orientation.x,msg.pose.pose.orientation.y,msg.pose.pose.orientation.z,msg.pose.pose.orientation.w)
        _,_,self.vehicle_yaw=euler_from_quaternion(amcl_quaternion)
        self.current_postion.x=msg.pose.pose.position.x
        self.current_postion.y=msg.pose.pose.position.y


if __name__ == '__main__':
    try:
        test_track=pure_pursuit()
    except rospy.ROSInterruptException:
        pass
```

END